Bike_Share_Analysis

February 20, 2018

1 2016 US Bike Share Activity Snapshot

1.1 Table of Contents

Introduction

> **Tip**: Quoted sections like this will provide helpful instructions on how to navigate and use a Jupyter notebook.

Over the past decade, bicycle-sharing systems have been growing in number and popularity in cities across the world. Bicycle-sharing systems allow users to rent bicycles for short trips, typically 30 minutes or less. Thanks to the rise in information technologies, it is easy for a user of the system to access a dock within the system to unlock or return bicycles. These technologies also provide a wealth of data that can be used to explore how these bike-sharing systems are used.

In this project, you will perform an exploratory analysis on data provided by Motivate, a bike-share system provider for many major cities in the United States. You will compare the system usage between three large cities: New York City, Chicago, and Washington, DC. You will also see if there are any differences within each system for those users that are registered, regular users and those users that are short-term, casual users.

Posing Questions

Before looking at the bike sharing data, you should start by asking questions you might want to understand about the bike share data. Consider, for example, if you were working for Motivate. What kinds of information would you want to know about in order to make smarter business decisions? If you were a user of the bike-share service, what factors might influence how you would want to use the service?

Question 1: Write at least two questions related to bike sharing that you think could be answered by data.

Answer: - 1. Which location in the city has maximum ridership? Which location has more customers compared to subscribers? - 2. Which location in the city has maximum trip duration?

> **Tip**: If you double click on this cell, you will see the text change so that all of the formatting is removed. This allows you to edit this block of text. This block of text is written using Markdown, which is a way to format text using headers, links, italics, and many other options using a plain-text syntax. You will also use Markdown later in the Nanodegree program. Use **Shift** + **Enter** or **Shift** + **Return** to run the cell and show its rendered form.

Data Collection and Wrangling

Now it's time to collect and explore our data. In this project, we will focus on the record of individual trips taken in 2016 from our selected cities: New York City, Chicago, and Washington, DC. Each of these cities has a page where we can freely download the trip data.:

- New York City (Citi Bike): Link
- Chicago (Divvy): Link
- Washington, DC (Capital Bikeshare): Link

If you visit these pages, you will notice that each city has a different way of delivering its data. Chicago updates with new data twice a year, Washington DC is quarterly, and New York City is monthly. **However, you do not need to download the data yourself.** The data has already been collected for you in the /data/ folder of the project files. While the original data for 2016 is spread among multiple files for each city, the files in the /data/ folder collect all of the trip data for the year into one file per city. Some data wrangling of inconsistencies in timestamp format within each city has already been performed for you. In addition, a random 2% sample of the original data is taken to make the exploration more manageable.

Question 2: However, there is still a lot of data for us to investigate, so it's a good idea to start off by looking at one entry from each of the cities we're going to analyze. Run the first code cell below to load some packages and functions that you'll be using in your analysis. Then, complete the second code cell to print out the first trip recorded from each of the cities (the second line of each data file).

> **Tip**: You can run a code cell like you formatted Markdown cells above by clicking on the cell and using the keyboard shortcut **Shift** + **Enter** or **Shift** + **Return**. Alternatively, a code cell can be executed using the **Play** button in the toolbar after selecting it. While the cell is running, you will see an asterisk in the message to the left of the cell, i.e. In [*]:. The asterisk will change into a number to show that execution has completed, e.g. In [1]. If there is output, it will show up as Out [1]:, with an appropriate number to match the "In" number.

```
In [1]: ## import all necessary packages and functions.
        import csv # read and write csv files
        from datetime import datetime,date # operations to parse dates
        from pprint import pprint # use to print data structures like dictionaries in
                                  # a nicer way than the base print function.
```

```
In [35]: def print_first_point(filename):
             """
             This function prints and returns the first data point (second row) from
             a csv file that includes a header row.
             """
             # print city name for reference
             city = filename.split('-')[0].split('/')[-1]
             print('\nCity: {}'.format(city))

             with open(filename, 'r') as f_in:
                 ## TODO: Use the csv library to set up a DictReader object. ##
                 ## see https://docs.python.org/3/library/csv.html          ##
                 trip_reader = csv.DictReader(f_in)

                 ## TODO: Use a function on the DictReader object to read the    ##
                 ## first trip from the data file and store it in a variable.    ##
                 ## see https://docs.python.org/3/library/csv.html#reader-objects ##
                 first_trip = trip_reader.__next__()

                 ## TODO: Use the pprint library to print the first trip. ##
                 ## see https://docs.python.org/3/library/pprint.html     ##
                 pprint(first_trip)

             # output city name and first trip for later testing
             return (city, first_trip)

         # list of files for each city
         data_files = ['./data/NYC-CitiBike-2016.csv',
                       './data/Chicago-Divvy-2016.csv',
                       './data/Washington-CapitalBikeshare-2016.csv',]

         # print the first trip from each file, store in dictionary
         example_trips = {}
         for data_file in data_files:
             city, first_trip = print_first_point(data_file)
             example_trips[city] = first_trip


City: NYC
OrderedDict([('tripduration', '839'),
             ('starttime', '1/1/2016 00:09:55'),
             ('stoptime', '1/1/2016 00:23:54'),
             ('start station id', '532'),
             ('start station name', 'S 5 Pl & S 4 St'),
             ('start station latitude', '40.710451'),
             ('start station longitude', '-73.960876'),
             ('end station id', '401'),
             ('end station name', 'Allen St & Rivington St'),
```

```
                    ('end station latitude', '40.72019576'),
                    ('end station longitude', '-73.98997825'),
                    ('bikeid', '17109'),
                    ('usertype', 'Customer'),
                    ('birth year', ''),
                    ('gender', '0')])


City: Chicago
OrderedDict([('trip_id', '9080545'),
                    ('starttime', '3/31/2016 23:30'),
                    ('stoptime', '3/31/2016 23:46'),
                    ('bikeid', '2295'),
                    ('tripduration', '926'),
                    ('from_station_id', '156'),
                    ('from_station_name', 'Clark St & Wellington Ave'),
                    ('to_station_id', '166'),
                    ('to_station_name', 'Ashland Ave & Wrightwood Ave'),
                    ('usertype', 'Subscriber'),
                    ('gender', 'Male'),
                    ('birthyear', '1990')])


City: Washington
OrderedDict([('Duration (ms)', '427387'),
                    ('Start date', '3/31/2016 22:57'),
                    ('End date', '3/31/2016 23:04'),
                    ('Start station number', '31602'),
                    ('Start station', 'Park Rd & Holmead Pl NW'),
                    ('End station number', '31207'),
                    ('End station', 'Georgia Ave and Fairmont St NW'),
                    ('Bike number', 'W20842'),
                    ('Member Type', 'Registered')])
```

If everything has been filled out correctly, you should see below the printout of each city name (which has been parsed from the data file name) that the first trip has been parsed in the form of a dictionary. When you set up a `DictReader` object, the first row of the data file is normally interpreted as column names. Every other row in the data file will use those column names as keys, as a dictionary is generated for each row.

This will be useful since we can refer to quantities by an easily-understandable label instead of just a numeric index. For example, if we have a trip stored in the variable `row`, then we would rather get the trip duration from `row['duration']` instead of `row[0]`.

Condensing the Trip Data

It should also be observable from the above printout that each city provides different information. Even where the information is the same, the column names and formats are sometimes different. To make things as simple as possible when we get to the actual exploration, we should trim and clean the data. Cleaning the data makes sure that the data formats across the cities are consistent, while trimming focuses only on the parts of the data we are most interested in to make the exploration easier to work with.

You will generate new data files with five values of interest for each trip: trip duration, starting month, starting hour, day of the week, and user type. Each of these may require additional wrangling depending on the city:

- **Duration**: This has been given to us in seconds (New York, Chicago) or milliseconds (Washington). A more natural unit of analysis will be if all the trip durations are given in terms of minutes.
- **Month**, **Hour**, **Day of Week**: Ridership volume is likely to change based on the season, time of day, and whether it is a weekday or weekend. Use the start time of the trip to obtain these values. The New York City data includes the seconds in their timestamps, while Washington and Chicago do not. The `datetime` package will be very useful here to make the needed conversions.
- **User Type**: It is possible that users who are subscribed to a bike-share system will have different patterns of use compared to users who only have temporary passes. Washington divides its users into two types: 'Registered' for users with annual, monthly, and other longer-term subscriptions, and 'Casual', for users with 24-hour, 3-day, and other short-term passes. The New York and Chicago data uses 'Subscriber' and 'Customer' for these groups, respectively. For consistency, you will convert the Washington labels to match the other two.

Question 3a: Complete the helper functions in the code cells below to address each of the cleaning tasks described above.

```
In [36]: def duration_in_mins(datum, city):
             """
             Takes as input a dictionary containing info about a single trip (datum) and
             its origin city (city) and returns the trip duration in units of minutes.

             Remember that Washington is in terms of milliseconds while Chicago and NYC
             are in terms of seconds.

             HINT: The csv module reads in all of the data as strings, including numeric
             values. You will need a function to convert the strings into an appropriate
             numeric type when making your transformations.
             see https://docs.python.org/3/library/functions.html
             """

             # YOUR CODE HERE
             if city == 'Washington' :
                 duration = int(datum['Duration (ms)'])/60000
             else :
                 duration = int(datum['tripduration'])/60

             return duration


         # Some tests to check that your code works. There should be no output if all of
         # the assertions pass. The `example_trips` dictionary was obtained from when
         # you printed the first trip from each of the original data files.
```

```
        tests = {'NYC': 13.9833,
                 'Chicago': 15.4333,
                 'Washington': 7.1231}

        for city in tests:
            assert abs(duration_in_mins(example_trips[city], city) - tests[city]) < .001

In [37]: def time_of_trip(datum, city):
             """
             Takes as input a dictionary containing info about a single trip (datum) and
             its origin city (city) and returns the month, hour, and day of the week in
             which the trip was made.

             Remember that NYC includes seconds, while Washington and Chicago do not.

             HINT: You should use the datetime module to parse the original date
             strings into a format that is useful for extracting the desired information.
             see https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior
             """

             # YOUR CODE HERE
             if city == 'NYC' :
                 starttime = datetime.strptime(datum['starttime'],"%m/%d/%Y %H:%M:%S")
                 month = int(starttime.strftime("%m"))
                 hour = int(starttime.strftime("%H"))
                 day_of_week = starttime.strftime("%A")
             elif city == 'Washington' :
                 starttime = datetime.strptime(datum['Start date'],"%m/%d/%Y %H:%M")
                 month = int(starttime.strftime("%m"))
                 hour = int(starttime.strftime("%H"))
                 day_of_week = starttime.strftime("%A")
             elif city == 'Chicago' :
                 starttime = datetime.strptime(datum['starttime'],"%m/%d/%Y %H:%M")
                 month = int(starttime.strftime("%m"))
                 hour = int(starttime.strftime("%H"))
                 day_of_week = starttime.strftime("%A")

             return (month, hour, day_of_week)


         # Some tests to check that your code works. There should be no output if all of
         # the assertions pass. The `example_trips` dictionary was obtained from when
         # you printed the first trip from each of the original data files.
         tests = {'NYC': (1, 0, 'Friday'),
                  'Chicago': (3, 23, 'Thursday'),
                  'Washington': (3, 22, 'Thursday')}

         for city in tests:
```

```
              assert time_of_trip(example_trips[city], city) == tests[city]

In [38]: def type_of_user(datum, city):
             """
             Takes as input a dictionary containing info about a single trip (datum) and
             its origin city (city) and returns the type of system user that made the
             trip.

             Remember that Washington has different category names compared to Chicago
             and NYC.
             """

             # YOUR CODE HERE
             if city == 'Washington' :
                 if datum['Member Type'] == 'Registered' :
                     user_type = 'Subscriber'
                 else :
                     user_type = 'Customer'
             else :
                 user_type = datum['usertype']

             return user_type


         # Some tests to check that your code works. There should be no output if all of
         # the assertions pass. The `example_trips` dictionary was obtained from when
         # you printed the first trip from each of the original data files.
         tests = {'NYC': 'Customer',
                  'Chicago': 'Subscriber',
                  'Washington': 'Subscriber'}

         for city in tests:
             assert type_of_user(example_trips[city], city) == tests[city]
```

Question 3b: Now, use the helper functions you wrote above to create a condensed data file for each city consisting only of the data fields indicated above. In the /examples/ folder, you will see an example datafile from the Bay Area Bike Share before and after conversion. Make sure that your output is formatted to be consistent with the example file.

```
In [56]: def condense_data(in_file, out_file, city):
             """
             This function takes full data from the specified input file
             and writes the condensed data to a specified output file. The city
             argument determines how the input file will be parsed.

             HINT: See the cell below to see how the arguments are structured!
             """
```

```python
            with open(out_file, 'w') as f_out, open(in_file, 'r') as f_in:
                # set up csv DictWriter object - writer requires column names for the
                # first row as the "fieldnames" argument
                out_colnames = ['duration', 'month', 'hour', 'day_of_week', 'user_type']
                trip_writer = csv.DictWriter(f_out, fieldnames = out_colnames)
                trip_writer.writeheader()

                ## TODO: set up csv DictReader object ##
                trip_reader = csv.DictReader(f_in)

                # collect data from and process each row
                for row in trip_reader:
                    # set up a dictionary to hold the values for the cleaned and trimmed
                    # data point
                    new_point = {}

                    ## TODO: use the helper functions to get the cleaned data from   ##
                    ## the original data dictionaries.                               ##
                    ## Note that the keys for the new_point dictionary should match ##
                    ## the column names set in the DictWriter object above.          ##
                    new_point['duration'] = duration_in_mins(row, city)
                    new_point['month'] = time_of_trip(row, city)[0]
                    new_point['hour'] = time_of_trip(row, city)[1]
                    new_point['day_of_week'] = time_of_trip(row, city)[2]
                    new_point['user_type'] = type_of_user(row, city)


                    ## TODO: write the processed information to the output file.      ##
                    ## see https://docs.python.org/3/library/csv.html#writer-objects ##
                    trip_writer.writerow(new_point)


In [57]: # Run this cell to check your work
         city_info = {'Washington': {'in_file': './data/Washington-CapitalBikeshare-2016.csv',
                                     'out_file': './data/Washington-2016-Summary.csv'},
                      'Chicago': {'in_file': './data/Chicago-Divvy-2016.csv',
                                  'out_file': './data/Chicago-2016-Summary.csv'},
                      'NYC': {'in_file': './data/NYC-CitiBike-2016.csv',
                              'out_file': './data/NYC-2016-Summary.csv'}}

         for city, filenames in city_info.items():
             condense_data(filenames['in_file'], filenames['out_file'], city)
             print_first_point(filenames['out_file'])


City: Washington
OrderedDict([('duration', '7.123116666666666'),
             ('month', '3'),
```

```
            ('hour', '22'),
            ('day_of_week', 'Thursday'),
            ('user_type', 'Subscriber')])

City: Chicago
OrderedDict([('duration', '15.433333333333334'),
            ('month', '3'),
            ('hour', '23'),
            ('day_of_week', 'Thursday'),
            ('user_type', 'Subscriber')])

City: NYC
OrderedDict([('duration', '13.983333333333333'),
            ('month', '1'),
            ('hour', '0'),
            ('day_of_week', 'Friday'),
            ('user_type', 'Customer')])
```

> **Tip**: If you save a jupyter Notebook, the output from running code blocks will also be saved. However, the state of your workspace will be reset once a new session is started. Make sure that you run all of the necessary code blocks from your previous session to reestablish variables and functions before picking up where you last left off.

Exploratory Data Analysis

Now that you have the data collected and wrangled, you're ready to start exploring the data. In this section you will write some code to compute descriptive statistics from the data. You will also be introduced to the `matplotlib` library to create some basic histograms of the data.

Statistics

First, let's compute some basic counts. The first cell below contains a function that uses the csv module to iterate through a provided data file, returning the number of trips made by subscribers and customers. The second cell runs this function on the example Bay Area data in the `/examples/` folder. Modify the cells to answer the question below.

Question 4a: Which city has the highest number of trips? Which city has the highest proportion of trips made by subscribers? Which city has the highest proportion of trips made by short-term customers?

Answer: New York City has highest proportion of trips made by subscribers, highest proportion of trips made by short term customers and highest number of trips

```python
In [58]: def number_of_trips(filename):
             """
             This function reads in a file with trip data and reports the number of
             trips made by subscribers, customers, and total overall.
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)
```

9

```python
                # initialize count variables
                n_subscribers = 0
                n_customers = 0

                # tally up ride types
                for row in reader:
                    if row['user_type'] == 'Subscriber':
                        n_subscribers += 1
                    else:
                        n_customers += 1

                # compute total number of rides
                n_total = n_subscribers + n_customers

                # return tallies as a tuple
                return(n_subscribers, n_customers, n_total)
```

In [59]:
```python
## Modify this and the previous cell to answer Question 4a. Remember to run ##
## the function on the cleaned data files you created from Question 3.      ##

# set up a dictionary to hold the values for the trips by subsciber,
# trips by customer and total trips
city_total = {}
city_subscribers = {}
city_customers = {}

# use the helper function to get the trips details for each city
for city, filenames in city_info.items():
    city_total[city] = number_of_trips(filenames['out_file'])[2]
    city_subscribers[city] = number_of_trips(filenames['out_file'])[0]
    city_customers[city] = number_of_trips(filenames['out_file'])[1]

# Find the city having maximum number of trips usign max function
max_total = max(city_total, key=city_total.get)
max_subscribers = max(city_subscribers, key=city_subscribers.get)
max_customers = max(city_customers, key=city_customers.get)

# Print the trips details and also the city having maximum trips
print(f'City->Total->Subscribers->Customers')
for keys in city_total :
    print(f'{keys}->{city_total[keys]}->{city_subscribers[keys]}->{city_customers[keys]

print(f'\n{max_total} has highest number of trips')
print(f'{max_subscribers} has highest number of trips by subscribers')
print(f'{max_customers} has highest number of trips by customers')
```

```
City->Total->Subscribers->Customers
Washington->66326->51753->14573
```

```
Chicago->72131->54982->17149
NYC->276798->245896->30902

NYC has highest number of trips
NYC has highest number of trips by subscribers
NYC has highest number of trips by customers
```

Tip: In order to add additional cells to a notebook, you can use the "Insert Cell Above" and "Insert Cell Below" options from the menu bar above. There is also an icon in the toolbar for adding new cells, with additional icons for moving the cells up and down the document. By default, new cells are of the code type; you can also specify the cell type (e.g. Code or Markdown) of selected cells from the Cell menu or the dropdown in the toolbar.

Now, you will write your own code to continue investigating properties of the data.

Question 4b: Bike-share systems are designed for riders to take short trips. Most of the time, users are allowed to take trips of 30 minutes or less with no additional charges, with overage charges made for trips of longer than that duration. What is the average trip length for each city? What proportion of rides made in each city are longer than 30 minutes?

Answer: It is tabulated as follows

City
Average Trip duration(min)
Proportion of trips longer than 30 minutes(%)
Washington
18.93
10.84
Chicago
16.56
8.33
New York City
15.81
7.3

```
In [60]:  ## Use this and additional cells to answer Question 4b.        ##
          ##                                                             ##
          ## HINT: The csv module reads in all of the data as strings, including  ##
          ## numeric values. You will need a function to convert the strings     ##
          ## into an appropriate numeric type before you aggregate data.         ##
          ## TIP: For the Bay Area example, the average trip length is 14 minutes ##
          ## and 3.5% of trips are longer than 30 minutes.                ##

          def duration_of_trips(filename):
              """
              This function reads in a file with trip data and reports the duration of
              trips made by users.
              """
              with open(filename, 'r') as f_in:
```

11

```python
                    # set up csv reader object
                    reader = csv.DictReader(f_in)

                    # initialize count variables
                    sub_trips = 0
                    cus_trips = 0
                    tot_trips = 0

                    # compute total duration of all trips for subscriber#
                    # and customer                                        #
                    for row in reader :
                        if row['user_type'] == 'Subscriber' :
                            sub_trips += float(row['duration'])
                        else :
                            cus_trips += float(row['duration'])

                    # Calculate Total trip duration                       #

                    tot_trips = sub_trips + cus_trips

                    # return tallies as a tuple
                    return(sub_trips, cus_trips, tot_trips)
```

In [61]:
```python
## Function to get trips longer than 30 minutes ##
def trips_longer_30(filename):
    """
    This function reads in a file with trip data and reports the duration of
    trips made by users.
    """
    with open(filename, 'r') as f_in:
        # set up csv reader object
        reader = csv.DictReader(f_in)

        # initialize count variables
        sub_trips = 0
        cus_trips = 0
        tot_trips = 0

        # compute number of rides longer than 30 minutes and total duration #
        # of all trips                                                       #
        for row in reader :
            if float(row['duration']) > 30 :
                if row['user_type'] == 'Subscriber':
                    sub_trips += 1
                else :
                    cus_trips += 1

        # compute total number of rides
```

```
            tot_trips = sub_trips + cus_trips

            # return tallies as a tuple
            return(sub_trips,cus_trips, tot_trips)
```

In [62]:
```
## Load the trip data to above function and get output ##

# set up a dictionary to hold the values for the average trip duration,
# proportion of rides longer than 30 minutes
city_average = {}
city_proportion = {}

# use the helper function to get the trips details for each city
city_average['Bay Area'] = duration_of_trips('./examples/BayArea-Y3-Summary.csv')[2]/nu
city_proportion['Bay Area'] = trips_longer_30('./examples/BayArea-Y3-Summary.csv')[2]*1

for city, filenames in city_info.items():
    city_average[city] = duration_of_trips(filenames['out_file'])[2]/city_total[city]
    city_proportion[city] = trips_longer_30(filenames['out_file'])[2]*100/city_total[ci

# Print the average duration and proportion of trips > 30 minutes #
# for each city                                                    #
print(f'City->Average duration->Proportion of trips longer 30 minutes')
for keys in city_average :
    print(f'{keys}->{city_average[keys]}->{city_proportion[keys]}')
```

```
City->Average duration->Proportion of trips longer 30 minutes
Bay Area->14.038656929671376->3.5243689474519764
Washington->18.93287355913725->10.83888671109369
Chicago->16.563629368787286->8.332062497400562
NYC->15.812592998022684->7.3024371563378345
```

Question 4c: Dig deeper into the question of trip duration based on ridership. Choose one city. Within that city, which type of user takes longer rides on average: Subscribers or Customers?

Answer: For New York City, Customers take longer rides on average - 19 min. longer

In [63]:
```
## Use this and additional cells to answer Question 4c. If you have    ##
## not done so yet, consider revising some of your previous code to     ##
## make use of functions for reusability.                              ##
##                                                                      ##
## TIP: For the Bay Area example data, you should find the average      ##
## Subscriber trip duration to be 9.5 minutes and the average Customer ##
## trip duration to be 54.6 minutes. Do the other cities have this      ##
## level of difference?                                                ##
## Load the trip data to above function and get output ##

# set up a dictionary to hold the values for the average trip duration,
city_subaverage = {}
```

```
        city_cusaverage = {}

        # use the helper function to get the trips details for each city
        city_subaverage['Bay Area'] = duration_of_trips('./examples/BayArea-Y3-Summary.csv')[0]
        city_cusaverage['Bay Area'] = duration_of_trips('./examples/BayArea-Y3-Summary.csv')[1]

        for city, filenames in city_info.items():
            city_subaverage[city] = duration_of_trips(filenames['out_file'])[0]/city_subscriber
            city_cusaverage[city] = duration_of_trips(filenames['out_file'])[1]/city_customers[

        # Print the average Subscriber and Customer trip #
        # duration for each city                          #
        print(f'City->Average Subscriber trip duration->Average Customer trip duration')
        for keys in city_average :
            print(f'{keys}->{city_subaverage[keys]}->{city_cusaverage[keys]}')

City->Average Subscriber trip duration->Average Customer trip duration
Bay Area->9.512633839275217->54.55121116377032
Washington->12.528120499294745->41.67803139252976
Chicago->12.067201690250076->30.979781133982506
NYC->13.680790523907177->32.77595139473187
```

Visualizations
The last set of values that you computed should have pulled up an interesting result. While the mean trip time for Subscribers is well under 30 minutes, the mean trip time for Customers is actually *above* 30 minutes! It will be interesting for us to look at how the trip times are distributed. In order to do this, a new library will be introduced here, `matplotlib`. Run the cell below to load the library and to generate an example plot.

```
In [64]: # load library
         import matplotlib.pyplot as plt

         # this is a 'magic word' that allows for plots to be displayed
         # inline with the notebook. If you want to know more, see:
         # http://ipython.readthedocs.io/en/stable/interactive/magics.html
         %matplotlib inline

         # example histogram, data taken from bay area sample
         data = [ 7.65,  8.92,  7.42,  5.50, 16.17,  4.20,  8.98,  9.62, 11.48, 14.33,
                 19.02, 21.53,  3.90,  7.97,  2.62,  2.67,  3.08, 14.40, 12.90,  7.83,
                 25.12,  8.30,  4.93, 12.43, 10.60,  6.17, 10.88,  4.78, 15.15,  3.53,
                  9.43, 13.32, 11.72,  9.85,  5.22, 15.10,  3.95,  3.17,  8.78,  1.88,
                  4.55, 12.68, 12.38,  9.78,  7.63,  6.45, 17.38, 11.90, 11.52,  8.63,]
         plt.hist(data)
         plt.title('Distribution of Trip Durations')
         plt.xlabel('Duration (m)')
         plt.show()
```

14

Distribution of Trip Durations
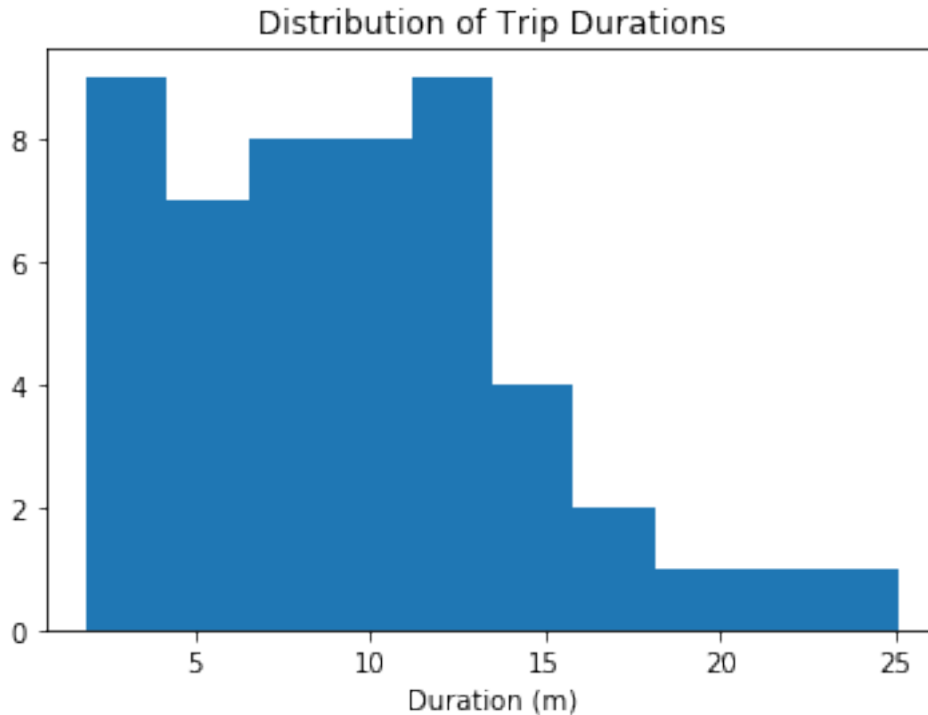
In the above cell, we collected fifty trip times in a list, and passed this list as the first argument to the `.hist()` function. This function performs the computations and creates plotting objects for generating a histogram, but the plot is actually not rendered until the `.show()` function is executed. The `.title()` and `.xlabel()` functions provide some labeling for plot context.

You will now use these functions to create a histogram of the trip times for the city you selected in question 4c. Don't separate the Subscribers and Customers for now: just collect all of the trip times and plot them.

```
In [65]: ## Use this and additional cells to collect all of the trip times as a list ##
         ## and then use pyplot functions to generate a histogram of trip times.      ##

         with open('./data/NYC-2016-Summary.csv', 'r') as f_in:
             # set up csv reader object
             reader = csv.DictReader(f_in)

             #Initialize values
             trip_data = []

             for row in reader :
                 trip_data.append(round(float(row['duration'])))

         #Plot the graph
         plt.hist(trip_data)
         plt.title('Distribution of Trip Durations')
         plt.xlabel('Duration (m)')
```

15

```
plt.show()
```


Distribution of Trip Durations
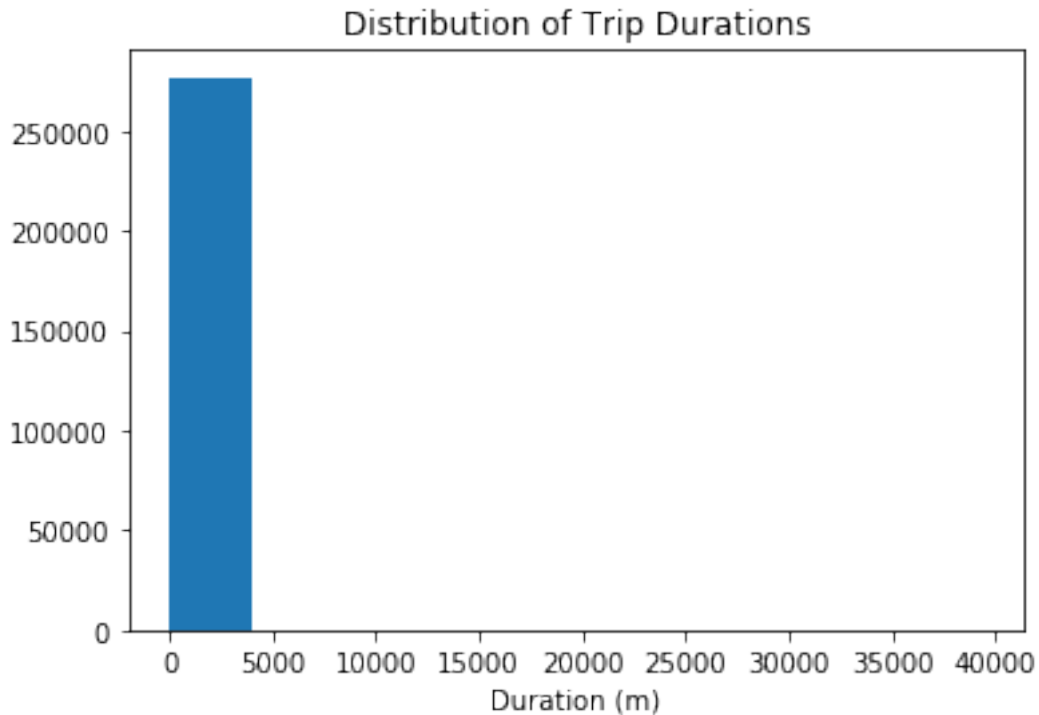
If you followed the use of the `.hist()` and `.show()` functions exactly like in the example, you're probably looking at a plot that's completely unexpected. The plot consists of one extremely tall bar on the left, maybe a very short second bar, and a whole lot of empty space in the center and right. Take a look at the duration values on the x-axis. This suggests that there are some highly infrequent outliers in the data. Instead of reprocessing the data, you will use additional parameters with the `.hist()` function to limit the range of data that is plotted. Documentation for the function can be found [here].

Question 5: Use the parameters of the `.hist()` function to plot the distribution of trip times for the Subscribers in your selected city. Do the same thing for only the Customers. Add limits to the plots so that only trips of duration less than 75 minutes are plotted. As a bonus, set the plots up so that bars are in five-minute wide intervals. For each group, where is the peak of each distribution? How would you describe the shape of each distribution?

Answer: For Subscriber group, peak is at [5-10]min interval and shape is Right skewed distribution For Customer group, peak is at [20-25] min interval and shape is Right skewed distribution

```
In [66]:  ## Use this and additional cells to answer Question 5. ##

          # load library
          import matplotlib.pyplot as plt

          # this is a 'magic word' that allows for plots to be displayed
          # inline with the notebook. If you want to know more, see:
```

16

```python
# http://ipython.readthedocs.io/en/stable/interactive/magics.html
%matplotlib inline

with open(city_info['NYC']['out_file'], 'r') as f_in:
    # set up csv reader object
    reader = csv.DictReader(f_in)

    #Initialize values
    sub_data = []
    cus_data = []

    for row in reader :
        if row['user_type'] == 'Subscriber':
            sub_data.append(round(float(row['duration'])))
        else :
            cus_data.append(round(float(row['duration'])))

#Plot the graph
plt.hist(sub_data, edgecolor='black', bins=[5*i for i in range(0,16)])
plt.title('Distribution of Subscriber Trip Durations')
plt.xlim(0,75)
plt.xlabel('Duration (m)')
plt.show()
plt.hist(cus_data, edgecolor='black', bins=[5*i for i in range(0,16)])
plt.xlim(0,75)
plt.title('Distribution of Customer Trip Durations')
plt.xlabel('Duration (m)')
plt.show()
```
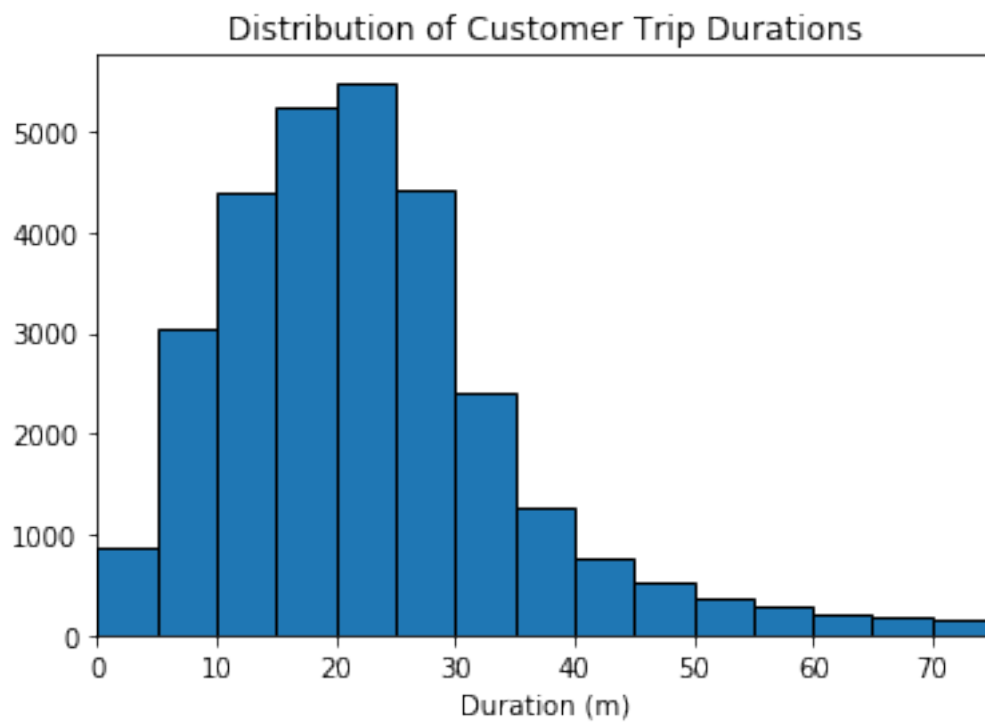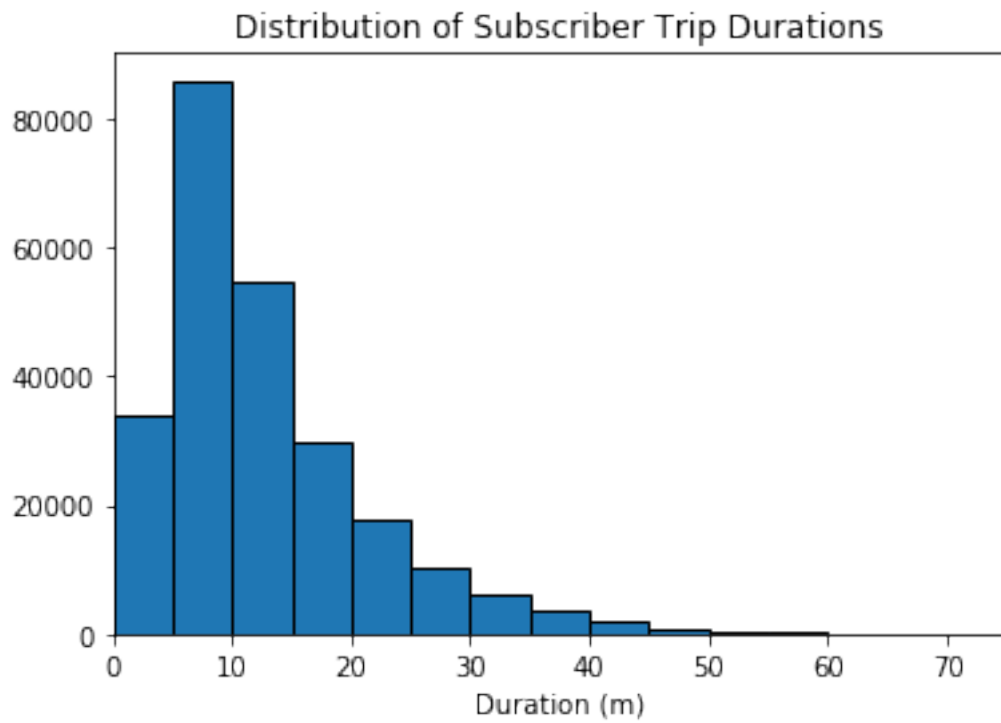
Distribution of Subscriber Trip Durations

Distribution of Customer Trip Durations

Performing Your Own Analysis

So far, you've performed an initial exploration into the data available. You have compared the relative volume of trips made between three U.S. cities and the ratio of trips made by Subscribers and Customers. For one of these cities, you have investigated differences between Subscribers and Customers in terms of how long a typical trip lasts. Now it is your turn to continue the exploration in a direction that you choose. Here are a few suggestions for questions to explore:

- How does ridership differ by month or season? Which month / season has the highest ridership? Does the ratio of Subscriber trips to Customer trips change depending on the month or season?
- Is the pattern of ridership different on the weekends versus weekdays? On what days are Subscribers most likely to use the system? What about Customers? Does the average duration of rides change depending on the day of the week?
- During what time of day is the system used the most? Is there a difference in usage patterns for Subscribers and Customers?

If any of the questions you posed in your answer to question 1 align with the bullet points above, this is a good opportunity to investigate one of them. As part of your investigation, you will need to create a visualization. If you want to create something other than a histogram, then you might want to consult the Pyplot documentation. In particular, if you are plotting values across a categorical variable (e.g. city, user type), a bar chart will be useful. The documentation page for .bar() includes links at the bottom of the page with examples for you to build off of for your own use.

Question 6: Continue the investigation by exploring another question that could be answered by the data available. Document the question you want to explore below. Your investigation should involve at least two variables and should compare at least two groups. You should also use at least one visualization as part of your explorations.

Answer: - How does ridership differ by month or season? Which month / season has the highest ridership? Does the ratio of Subscriber trips to Customer trips change depending on the month or season? - **Answer**: Monthly and Seasonal ridership plots are shown below. The system is having highest ridership in Fall season especially on September. During winter season, there is almost negligible usage by Customers causing Subscriber to Customer ratio high. Customer and Subscriber usage is high during Summer and Fall seasons. Customer ridership is more on August and Subscriber ridership is more on September

```
In [5]:  ## Use this and additional cells to continue to explore the dataset. ##
         ## Once you have performed your exploration, document your findings   ##
         ## in the Markdown cell above.                                        ##
         def mdnumber_of_trips(filename, comp, actual):
             """
             This function reads in a file with trip data and reports the duration of
             trips made by users.
             """
             with open(filename, 'r') as f_in:
                 # set up csv reader object
                 reader = csv.DictReader(f_in)

                 # initialize count variables
```

```
              sub_trips = 0
              cus_trips = 0
              tot_trips = 0

              # compute total duration of all trips for subscriber#
              # and customer                                        #
              for row in reader :
                  if row[comp] == actual:
                      if row['user_type'] == 'Subscriber' :
                          sub_trips += 1
                      else :
                          cus_trips += 1

              # Calculate Total trip duration                      #

              tot_trips = sub_trips + cus_trips

              # return tallies as a tuple
              return(sub_trips, cus_trips, tot_trips)

In [68]: ##Get ridership pattern by month/season    ##

         #Initialize dictionaries

         sub_mondata = []
         cus_mondata = []
         sub_sesdata = []
         cus_sesdata = []
         months = []
         rat_mondata = []

         #Get monthly trip data for all users ##
         for month in range(1,13):
             sub_mondata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'month', str(mon
             cus_mondata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'month', str(mon
             months.append(date(2018, month, 1).strftime('%b'))
             rat_mondata.append(sub_mondata[month-1]/cus_mondata[month-1])

         for i in range(1,4) :
             sub_sesdata.append(sub_mondata[3*i-1] + sub_mondata[3*i] + sub_mondata[3*i+1])
             cus_sesdata.append(cus_mondata[3*i-1] + cus_mondata[3*i] + cus_mondata[3*i+1])

         sub_sesdata.append(sub_mondata[11] + sub_mondata[0] + sub_mondata[1])
         cus_sesdata.append(cus_mondata[11] + cus_mondata[0] + cus_mondata[1])

In [69]: ##Plot the graph ##
         ind = [i for i in range(1,13)]
         p1 = plt.bar(ind, sub_mondata, edgecolor='black', color='red')
```
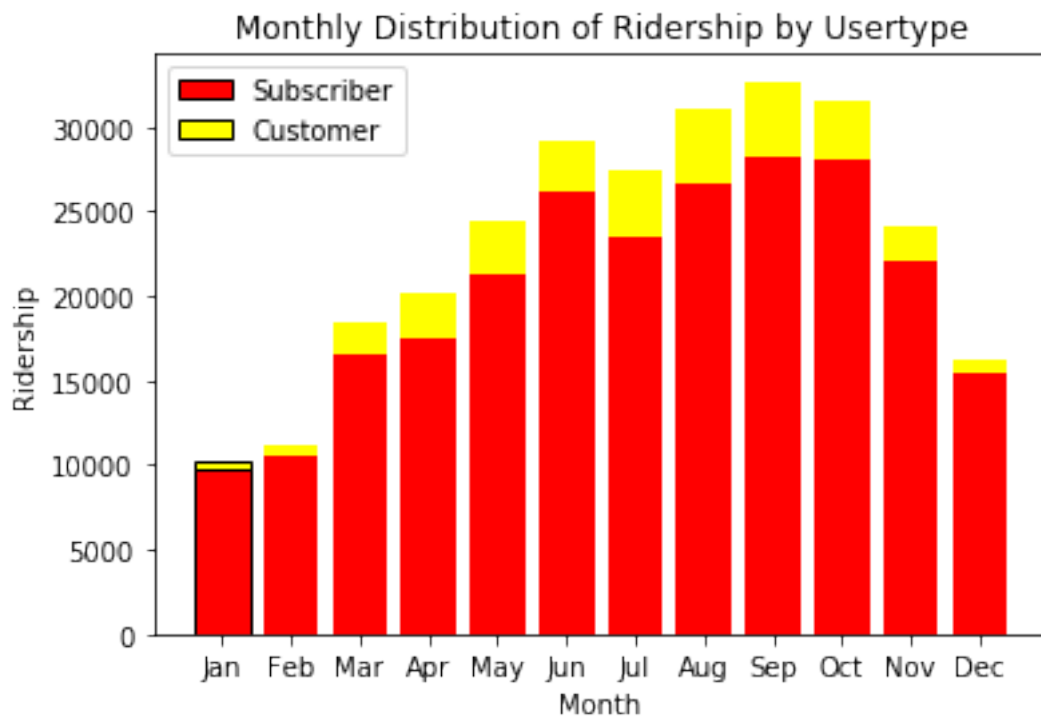
20

```
p2 = plt.bar(ind, cus_mondata, edgecolor='black', color='yellow', bottom=sub_mondata)
plt.title('Monthly Distribution of Ridership by Usertype')
plt.xticks(ind, months)
plt.xlabel('Month')
plt.ylabel('Ridership')
plt.legend((p1[0], p2[0]), ('Subscriber', 'Customer'))
plt.show()

print(f'Customer ridership is more on {months[cus_mondata.index(max(cus_mondata))]}')
print(f'Subscriber ridership is more on {months[sub_mondata.index(max(sub_mondata))]}')
print(f'Subscriber to Customer ridership ratio is more on {months[rat_mondata.index(max

p1 = plt.bar(ind, rat_mondata, edgecolor='black')
plt.title('Monthly Distribution of Ridership ratio')
plt.xticks(ind, months)
plt.xlabel('Month')
plt.ylabel('Ridership ratio')
plt.show()
```
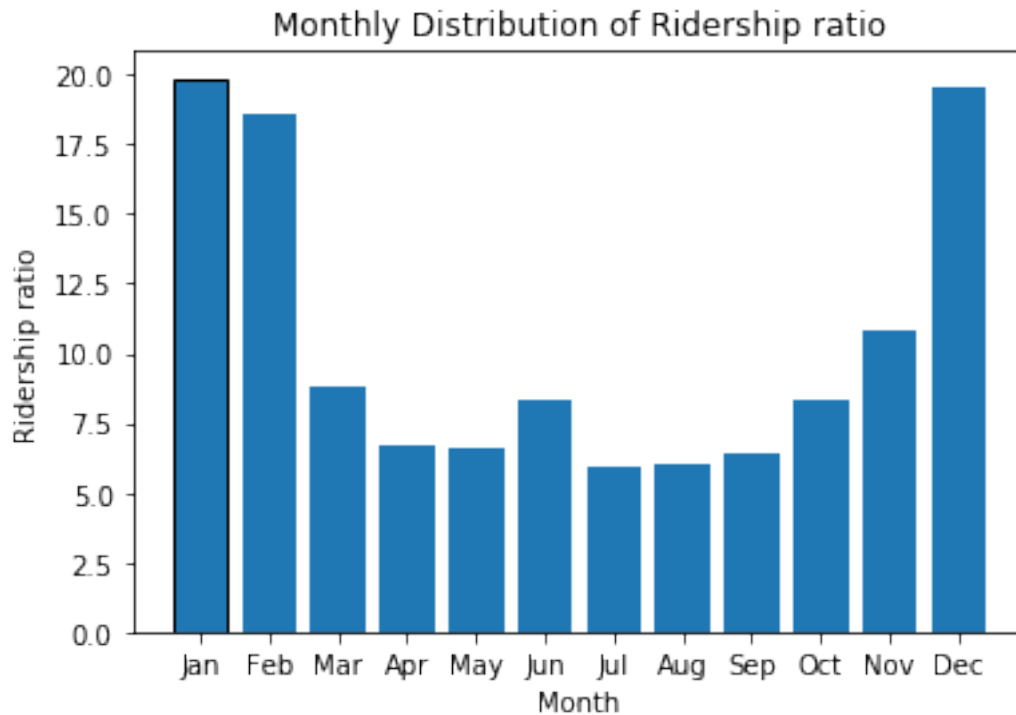


```
Customer ridership is more on Aug
Subscriber ridership is more on Sep
Subscriber to Customer ridership ratio is more on Jan
```

Monthly Distribution of Ridership ratio

In [70]: *#Plot the graph*
```
ses = [i for i in range(1,5)]
p1 = plt.bar(ses, sub_sesdata, edgecolor='black', color='red')
p2 = plt.bar(ses, cus_sesdata, edgecolor='black', color='yellow', bottom=sub_sesdata)
plt.title('Seasonal Distribution of Ridership by Usertype')
plt.xticks(ses, ['Spring', 'Summer', 'Fall', 'Winter'])
plt.xlabel('Season')
plt.ylabel('Ridership')
plt.legend((p1[0], p2[0]), ('Subscriber', 'Customer'))
plt.show()

print(f'Customer ridership is more on {ses[cus_sesdata.index(max(cus_sesdata))]}')
print(f'Subscriber ridership is more on {ses[sub_sesdata.index(max(sub_sesdata))]}')
```

Seasonal Distribution of Ridership by Usertype
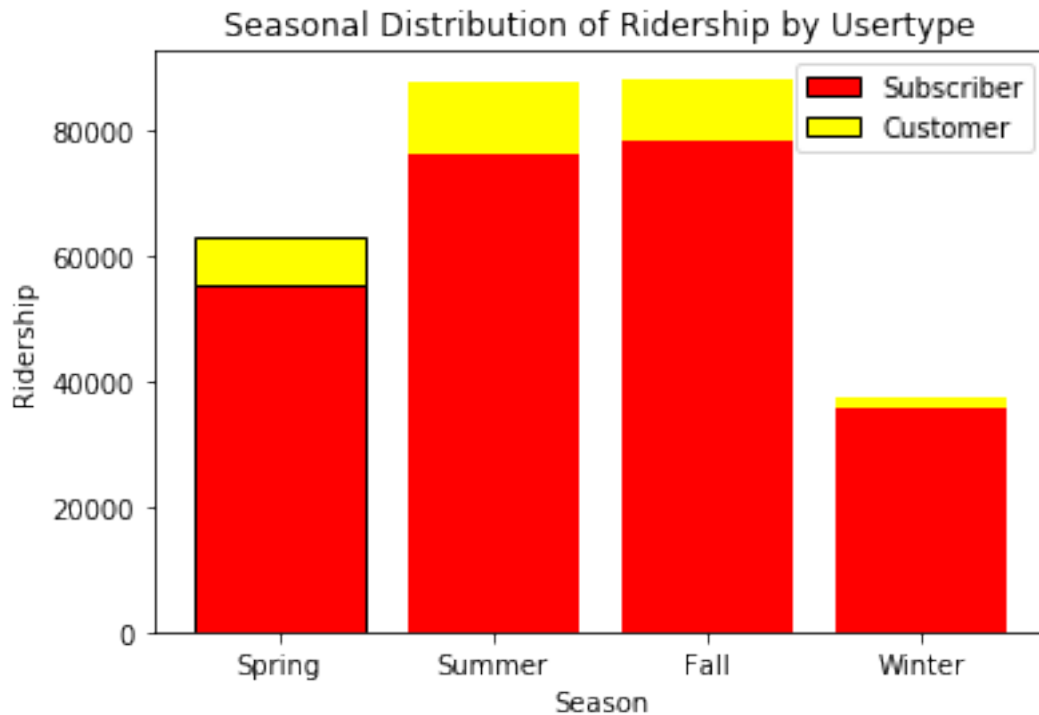

```
Customer ridership is more on 2
Subscriber ridership is more on 3
```

- Is the pattern of ridership different on the weekends versus weekdays? On what days are Subscribers most likely to use the system? What about Customers? Does the average duration of rides change depending on the day of the week?

- **Answer**:Ridership pattern is different on weekends and weekdays Subscribers most likely use the system on Weekdays and Customers on Weekends. Average duration of rides depend on day of week and is usually more on weekends

```
In [71]:  ## Use this and additional cells to answer Question 4b.          ##
          ##                                                                ##
          ## HINT: The csv module reads in all of the data as strings, including  ##
          ## numeric values. You will need a function to convert the strings      ##
          ## into an appropriate numeric type before you aggregate data.          ##
          ## TIP: For the Bay Area example, the average trip length is 14 minutes ##
          ## and 3.5% of trips are longer than 30 minutes.                        ##

          def dduration_of_trips(filename, day):
              """
              This function reads in a file with trip data and reports the duration of
              trips made by users.
```

```python
        """
        with open(filename, 'r') as f_in:
            # set up csv reader object
            reader = csv.DictReader(f_in)

            # initialize count variables
            sub_trips = 0
            cus_trips = 0
            tot_trips = 0

            # compute total duration of all trips for subscriber#
            # and customer                                       #
            for row in reader :
                if row['day_of_week'] == day :
                    if row['user_type'] == 'Subscriber' :
                        sub_trips += float(row['duration'])
                    else :
                        cus_trips += float(row['duration'])

            # Calculate Total trip duration                      #

            tot_trips = sub_trips + cus_trips

            # return tallies as a tuple
            return(sub_trips, cus_trips, tot_trips)
```

In [78]: ##Get ridership pattern by day of week   ##

```python
#Initialize dictionaries

sub_daydata = []
cus_daydata = []
tot_daydata = []
sub_dayavg = []
cus_dayavg = []
tot_dayavg = []
days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']


#Get monthly trip data for all users ##
for day in range(1,8):
    sub_daydata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'day_of_week', d
    cus_daydata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'day_of_week', d
    tot_daydata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'day_of_week', d
    sub_dayavg.append(dduration_of_trips(city_info['NYC']['out_file'], days[day-1])[0]/
    cus_dayavg.append(dduration_of_trips(city_info['NYC']['out_file'], days[day-1])[1]/
    tot_dayavg.append(dduration_of_trips(city_info['NYC']['out_file'], days[day-1])[2]/
```

In [87]: ##Plot the graph ##

```
ind = [i for i in range(1,8)]
p1 = plt.bar(ind, sub_daydata, edgecolor='black', color='red')
p2 = plt.bar(ind, cus_daydata, edgecolor='black', color='yellow', bottom=sub_daydata)
plt.title('Daily Distribution of Ridership by Usertype')
plt.xticks(ind, days)
plt.xlabel('Day')
plt.ylabel('Ridership')
plt.legend((p1[0], p2[0]), ('Subscriber', 'Customer'))
plt.show()

print(f'Customer ridership is more on {days[cus_daydata.index(max(cus_daydata))]}')
print(f'Subscriber ridership is more on {days[sub_daydata.index(max(sub_daydata))]}')

plt.bar(ind, tot_dayavg, edgecolor='black', label='Total' )
plt.title('Daily Distribution of Ridership by Avg. duration')
plt.xticks(ind, days)
plt.xlabel('Day')
plt.ylabel('Ridership')
plt.show()
```
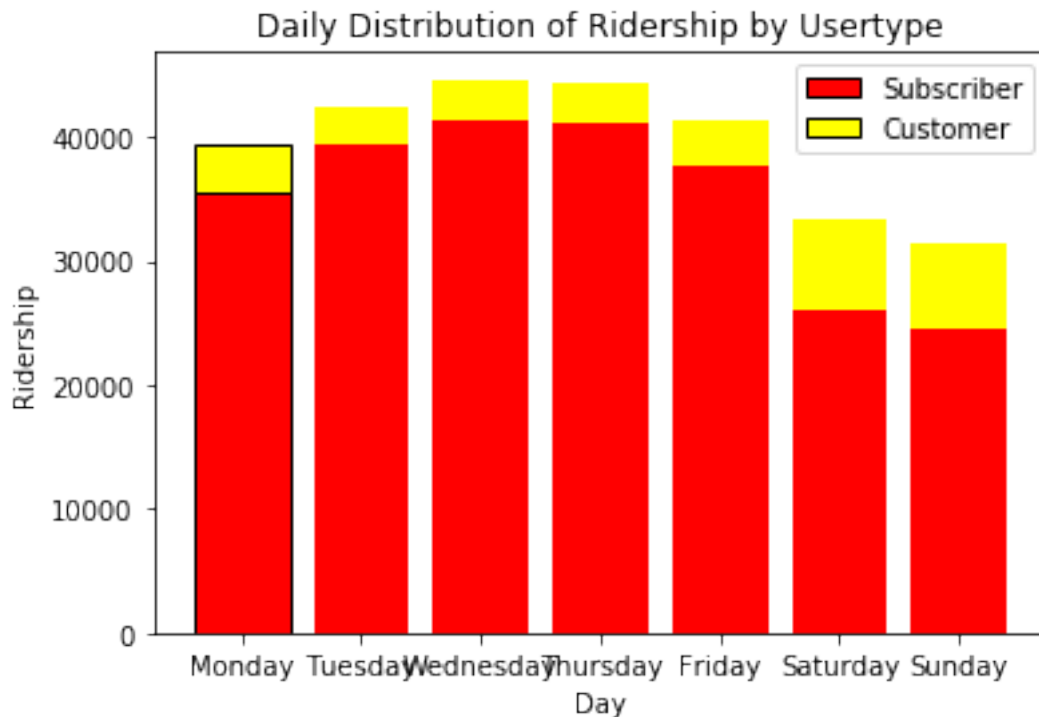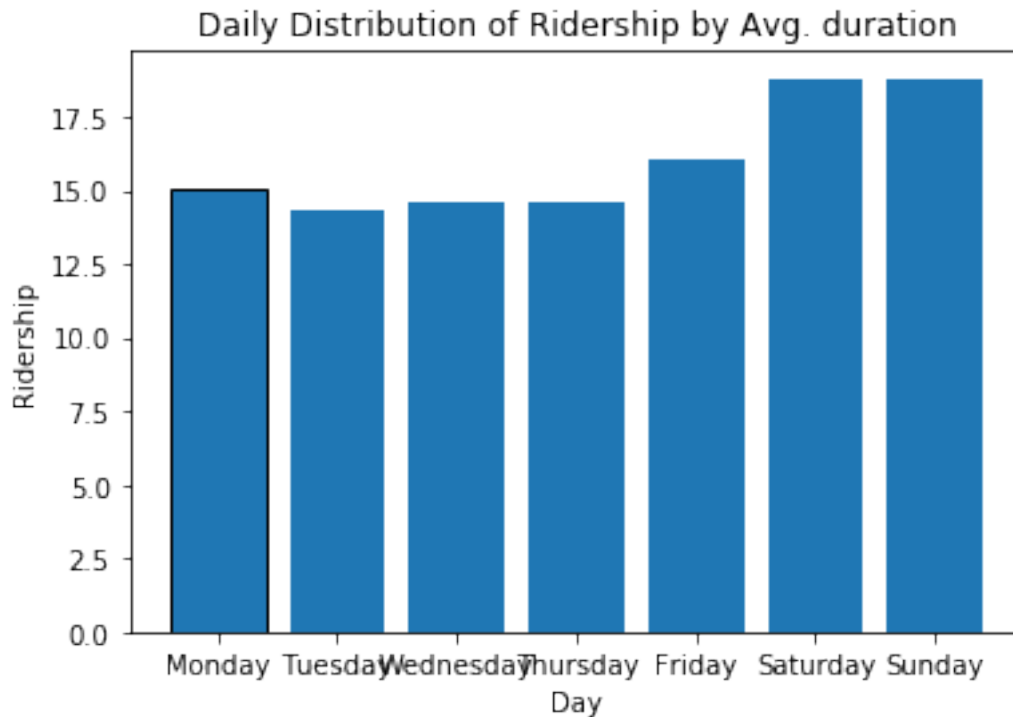


```
Customer ridership is more on Saturday
Subscriber ridership is more on Wednesday
```

Daily Distribution of Ridership by Avg. duration

- During what time of day is the system used the most? Is there a difference in usage patterns for Subscribers and Customers?
- **Answer** : System is used the most at 18'o clock. The usage pattern is different for subscribers and customers. Subscribers use the system most at 9-10'0 clock and 18-20'o clock which corresponds to office commute hours while customers use it the most at 11-18'0 clock corresponding to tourist hours

```
In [90]: ##Get ridership pattern by hour    ##

         #Initialize dictionaries

         sub_hourdata = []
         cus_hourdata = []
         tot_hourdata = []
         hours = [i for i in range(0,24)]


         #Get monthly trip data for all users ##
         for hour in range(0,24):
             sub_hourdata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'hour', str(hou
             cus_hourdata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'hour', str(hou
             tot_hourdata.append(mdnumber_of_trips(city_info['NYC']['out_file'], 'hour', str(hou

In [92]: ##Plot the graph ##
         p1 = plt.bar(hours, sub_hourdata, edgecolor='black', color='red')
```

```
p2 = plt.bar(hours, cus_hourdata, edgecolor='black', color='yellow', bottom=sub_hourdat
plt.title('Hourly Distribution of Ridership by Usertype')
plt.xlabel('Hour')
plt.ylabel('Ridership')
plt.legend((p1[0], p2[0]), ('Subscriber', 'Customer'))
plt.show()

print(f'Customer ridership is more on {cus_hourdata.index(max(cus_hourdata))}')
print(f'Subscriber ridership is more on {sub_hourdata.index(max(sub_hourdata))}')
```
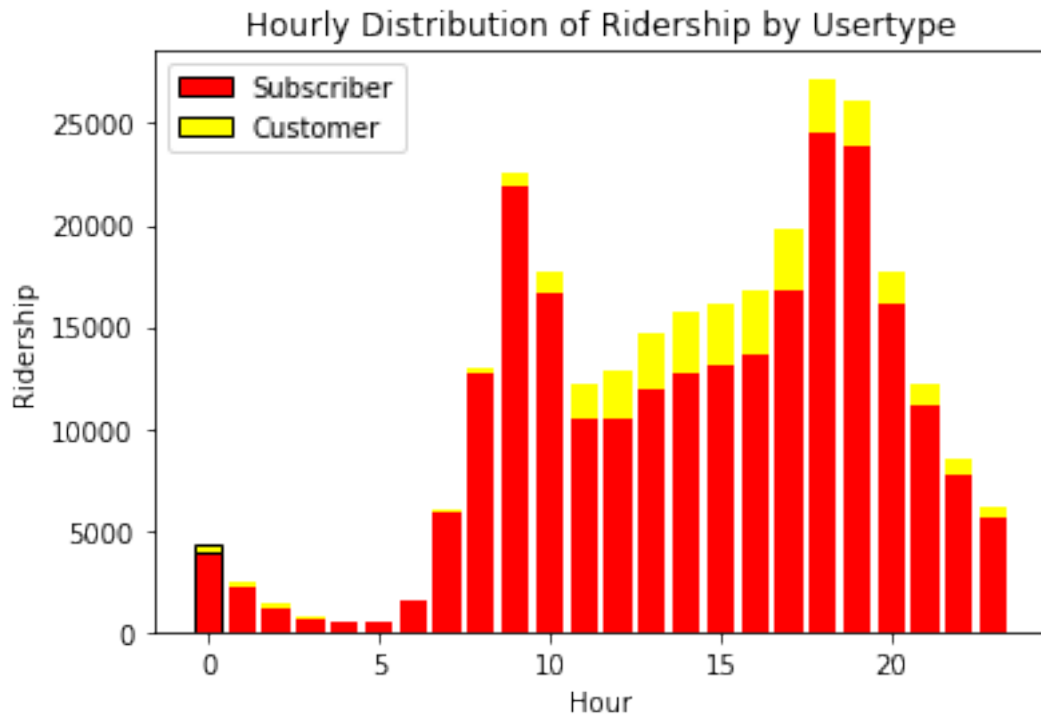


```
Customer ridership is more on 16
Subscriber ridership is more on 18
```

Conclusions

Congratulations on completing the project! This is only a sampling of the data analysis process: from generating questions, wrangling the data, and to exploring the data. Normally, at this point in the data analysis process, you might want to draw conclusions about the data by performing a statistical test or fitting the data to a model for making predictions. There are also a lot of potential analyses that could be performed on the data which are not possible with only the data provided. For example, detailed location data has not been investigated. Where are the most commonly used docks? What are the most common routes? As another example, weather has potential to have a large impact on daily ridership. How much is ridership impacted when there is rain or snow? Are subscribers or customers affected more by changes in weather?

Question 7: Putting the bike share data aside, think of a topic or field of interest where you would like to be able to apply the techniques of data science. What would you like to be able to learn from your chosen subject?

Answer: I would like to apply the techniques of datascience to analyze the fraud credit card transactions or SWIFT payments and develop a model to predict if a transaction is authentic or fraud one.

> **Tip**: If we want to share the results of our analysis with others, we aren't limited to giving them a copy of the jupyter Notebook (.ipynb) file. We can also export the Notebook output in a form that can be opened even for those without Python installed. From the **File** menu in the upper left, go to the **Download as** submenu. You can then choose a different format that can be viewed more generally, such as HTML (.html) or PDF (.pdf). You may need additional packages or software to perform these exports.

> If you are working on this project via the Project Notebook page in the classroom, you can also submit this project directly from the workspace. **Before you do that**, you should save an HTML copy of the completed project to the workspace by running the code cell below. If it worked correctly, the output code should be a 0, and if you click on the jupyter icon in the upper left, you should see your .html document in the workspace directory. Alternatively, you can download the .html copy of your report following the steps in the previous paragraph, then *upload* the report to the directory (by clicking the jupyter icon).

> Either way, once you've gotten the .html report in your workspace, you can complete your submission by clicking on the "Submit Project" button to the lower-right hand side of the workspace.

```
In [10]: from subprocess import call
         call(['python', '-m', 'nbconvert', 'Bike_Share_Analysis.ipynb'])

Out[10]: 0
```